22장 this

22.1 this 키워드

- 메서드는 자신이 속한 객체의 상태인 프로퍼티를 참조하고 변경할 수 있어야 한다.
- 이를 위해 자신이 혹산 객체를 가리키는 식별자를 참조할 수 있어야 한다.
- this는 자신이 속한 객체 또는 자신이 생성할 인스턴스를 가리키는 자기 참조 변수(self-referencing variable).
- this를 통해 자신이 속한 객체 또는 생성할 인스턴스의 프로퍼티나 메소드를 참조 가능.

▶ this 바인딩은 함수 호출 방식에 의해 동적으로 결정된다.

```
// 객체 리터럴
const circle = {
  radius: 5,
  getDiameter() {
    // this는 메서드를 호출한 객체를 가리킨다.
    return 2 * this.radius;
  }
};

console.log(circle.getDiameter()); // 10
```

```
// 생성자 함수
function Circle(radius) {
  // this는 생성자 함수가 생성할 인스턴스를 가리킨다.
  this.radius = radius;
}

Circle.prototype.getDiameter = function () {
  // this는 생성자 함수가 생성할 인스턴스를 가리킨다.
  return 2 * this.radius;
};

// 인스턴스 생성
```

```
const circle = new Circle(5);
console.log(circle.getDiameter()); // 10
```

```
// this는 어디서든지 참조 가능하다.
// 전역에서 this는 전역 객체 window를 가리킨다.
console.log(this); // window

function square(number) {
  // 일반 함수 내부에서 this는 전역 객체 window를 가리킨다.
  console.log(this); // window
  return number * number;
}
square(2);

const person = {
  name: 'Lee',
  getName() {
    // 메서드 내부에서 this는 메서드를 호출한 객체를 가리킨다.
    console.log(this); // {name: "Lee", getName: ƒ}
    return this.name;
  }
};
console.log(person.getName()); // Lee

function Person(name) {
  this.name = name;
  // 생성자 함수 내부에서 this는 생성자 함수가 생성할 인스턴스를 가리킨
다.
  console.log(this); // Person {name: "Lee"}
}

const me = new Person('Lee');
```

엄격 모드에서는 일반 함수 내부의 this에는 undefined가 바인딩된다.

22.2 함수 호출 방식과 this 바인딩

22.2.1 일반 함수 호출

this에는 기본적으로 전역 객체가 바인딩.

```
function foo() {
  console.log("foo's this: ", this);  // window
  function bar() {
    console.log("bar's this: ", this); // window
  }
  bar();
}
foo();
```

엄격 모드일 경우 undefined가 바인딩.

```
function foo() {
  'use strict';

  console.log("foo's this: ", this);  // undefined
  function bar() {
    console.log("bar's this: ", this); // undefined
  }
  bar();
}
foo();
```

```
// var 키워드로 선언한 전역 변수 value는 전역 객체의 프로퍼티다.
var value = 1;
// const 키워드로 선언한 전역 변수 value는 전역 객체의 프로퍼티가 아
니다.
// const value = 1;

const obj = {
  value: 100,
  foo() {
    console.log("foo's this: ", this);  // {value: 100, fo
o: f}
    console.log("foo's this.value: ", this.value); // 100
```

```
    // 메서드 내에서 정의한 중첩 함수
    function bar() {
      console.log("bar's this: ", this); // window
      console.log("bar's this.value: ", this.value); // 1
    }

    // 메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수
내부의 this에는 전역 객체가 바인딩된다.
    bar();
  }
};

obj.foo();
```

```
var value = 1;

const obj = {
  value: 100,
  foo() {
    console.log("foo's this: ", this); // {value: 100, foo:
f}
    // 콜백 함수 내부의 this에는 전역 객체가 바인딩된다.
    setTimeout(function () {
      console.log("callback's this: ", this); // window
      console.log("callback's this.value: ", this.value);
// 1
    }, 100);
  }
};

obj.foo();
```

- 중첩 함수 또는 콜백 함수를 일반함수로 사용하는 것은 문제가 생길 수 있다.
- this가 전역 객체를 참조하기 때문.
- 이를 해결할 수 있는 방법은 다음과 같다.

```
var value = 1;

const obj = {
  value: 100,
  foo() {
    // this 바인딩(obj)을 변수 that에 할당한다.
    const that = this;

    // 콜백 함수 내부에서 this 대신 that을 참조한다.
    setTimeout(function () {
      console.log(that.value); // 100
    }, 100);
  }
};

obj.foo();
```

```
var value = 1;

const obj = {
  value: 100,
  foo() {
    // 콜백 함수에 명시적으로 this를 바인딩한다.
    setTimeout(function () {
      console.log(this.value); // 100
    }.bind(this), 100);
  }
};

obj.foo();
```

또는 화살표 함수를 사용할 수 있다.

화살표 함수는 상위 스코프의 this를 가리킨다.

```
var value = 1;

const obj = {
```

```
  value: 100,
  foo() {
    // 화살표 함수 내부의 this는 상위 스코프의 this를 가리킨다.
    setTimeout(() => console.log(this.value), 100); // 100
  }
};

obj.foo();
```

22.2.2 메서드 호출

메서드 내부의 this는 메서드를 호출한 객체에 바인딩된다.

```
const person = {
  name: 'Lee',
  getName() {
    // 메서드 내부의 this는 메서드를 호출한 객체에 바인딩된다.
    return this.name;
  }
};

// 메서드 getName을 호출한 객체는 person이다.
console.log(person.getName()); // Lee
```

```
const anotherPerson = {
  name: 'Kim'
};
// getName 메서드를 anotherPerson 객체의 메서드로 할당
anotherPerson.getName = person.getName;

// getName 메서드를 호출한 객체는 anotherPerson이다.
console.log(anotherPerson.getName()); // Kim

// getName 메서드를 변수에 할당
const getName = person.getName;

// getName 메서드를 일반 함수로 호출
```

```
console.log(getName()); // ''
// 일반 함수로 호출된 getName 함수 내부의 this.name은 브라우저 환경에서 window.name과 같다.
// 브라우저 환경에서 window.name은 브라우저 창의 이름을 나타내는 빌트인 프로퍼티이며 기본값은 ''이다.
// Node.js 환경에서 this.name은 undefined다.
```

프로토타입 메서드 내부에서 사용된 this도 호출한 객체에 바인딩된다.

```
function Person(name) {
  this.name = name;
}

Person.prototype.getName = function () {
  return this.name;
};

const me = new Person('Lee');

// getName 메서드를 호출한 객체는 me다.
console.log(me.getName()); // ① Lee

Person.prototype.name = 'Kim';

// getName 메서드를 호출한 객체는 Person.prototype이다.
console.log(Person.prototype.getName()); // ② Kim
```

22.2.3 생성자 함수 호출

생성자 함수 내부의 this는 생성자 함수가 생성할 인스턴스에 바인딩된다.

```
// 생성자 함수
function Circle(radius) {
  // 생성자 함수 내부의 this는 생성자 함수가 생성할 인스턴스를 가리킨다.
  this.radius = radius;
  this.getDiameter = function () {
```

```
    return 2 * this.radius;
  };
}

// 반지름이 5인 Circle 객체를 생성
const circle1 = new Circle(5);
// 반지름이 10인 Circle 객체를 생성
const circle2 = new Circle(10);

console.log(circle1.getDiameter()); // 10
console.log(circle2.getDiameter()); // 20
```

```
// new 연산자와 함께 호출하지 않으면 생성자 함수로 동작하지 않는다. 즉, 일반적인 함수의 호출이다.
const circle3 = Circle(15);

// 일반 함수로 호출된 Circle에는 반환문이 없으므로 암묵적으로 undefined를 반환한다.
console.log(circle3); // undefined

// 일반 함수로 호출된 Circle 내부의 this는 전역 객체를 가리킨다.
console.log(radius); // 15
```

22.2.4 Function.prototype.apply / call / bind 메서드에 의한 간접 호출

- apply, call, bind는 Function.prototype의 메서드.
- 모든 함수가 상속받아 사용할 수 있다.

```
function getThisBinding() {
  return this;
}

// this로 사용할 객체
const thisArg = { a: 1 };
```

```
console.log(getThisBinding()); // window

// getThisBinding 함수를 호출하면서 인수로 전달한 객체를 getThisBi
nding 함수의 this에 바인딩한다.
console.log(getThisBinding.apply(thisArg)); // {a: 1}
console.log(getThisBinding.call(thisArg)); // {a: 1}
```

apply와 call 메서드는 함수를 호출하는 것.

첫 번째 인수로 전달한 객체를 호출한 함수의 this에 바인딩한다.

```
function getThisBinding() {
  console.log(arguments);
  return this;
}

// this로 사용할 객체
const thisArg = { a: 1 };

// getThisBinding 함수를 호출하면서 인수로 전달한 객체를 getThisBi
nding 함수의 this에 바인딩한다.
// apply 메서드는 호출할 함수의 인수를 배열로 묶어 전달한다.
console.log(getThisBinding.apply(thisArg, [1, 2, 3]));
// Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterato
r): ƒ]
// {a: 1}

// call 메서드는 호출할 함수의 인수를 쉼표로 구분한 리스트 형식으로 전
달한다.
console.log(getThisBinding.call(thisArg, 1, 2, 3));
// Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterato
r): ƒ]
// {a: 1}
```

apply는 인수를 배열로 묶어 전달한다.

call은 쉼표로 구분한 리스트 형식으로 전달한다.

그 외의 동작 방식은 apply와 call 모두 동일하다.

apply, call의 대표적인 용도는 유사 배열 객체에 배열 메서드를 사용하기 위한 경우.

```
function convertArgsToArray() {
  console.log(arguments);

  // arguments 객체를 배열로 변환
  // Array.prototype.slice를 인수없이 호출하면 배열의 복사본을 생성
한다.
  const arr = Array.prototype.slice.call(arguments);
  // const arr = Array.prototype.slice.apply(arguments);
  console.log(arr);

  return arr;
}

convertArgsToArray(1, 2, 3); // [1, 2, 3]
```

Function.prototype.bind는 this 바인딩이 교체된 새로운 함수를 생성해서 반환한다.

```
function getThisBinding() {
  return this;
}

// this로 사용할 객체
const thisArg = { a: 1 };

// bind 메서드는 첫 번째 인수로 전달한 thisArg로 this 바인딩이 교체
된
// getThisBinding 함수를 새롭게 생성해 반환한다.
console.log(getThisBinding.bind(thisArg)); // getThisBindin
g
// bind 메서드는 함수를 호출하지는 않으므로 명시적으로 호출해야 한다.
console.log(getThisBinding.bind(thisArg)()); // {a: 1}
```

메서드의 this와 메서드 내부의 중첩 함수 또는 콜백 함수의 this가 불일치하는 문제를 해결할 때 유용하다.

```
const person = {
  name: 'Lee',
  foo(callback) {
```

```
    // ① this => person
    setTimeout(callback, 100);
  }
};

person.foo(function () {
  console.log(`Hi! my name is ${this.name}.`); // ② Hi! my
name is undefined.
  // 일반 함수로 호출된 콜백 함수 내부의 this.name은 브라우저 환경에
서 window.name과 같다.
  // 브라우저 환경에서 window.name은 브라우저 창의 이름을 나타내는 빌
트인 프로퍼티이며 기본값은 ''이다.
  // Node.js 환경에서 this.name은 undefined다.
});
```

```
const person = {
  name: 'Lee',
  foo(callback) {
    // bind 메서드로 callback 함수 내부의 this 바인딩을 전달
    setTimeout(callback.bind(this), 100);
  }
};

person.foo(function () {
  console.log(`Hi! my name is ${this.name}.`); // Hi! my na
me is Lee.
});
```